Parámetros de prestaciones

En términos de velocidad.

- Tiempo de ejecución.
- Productividad (*throughput*)

Tiempo de ejecución

- En función de la duración del ciclo de reloj y del número de instrucciones:

$$T.ejecución = Nº\,ciclos \bullet Tciclo = \frac{Nº\,ciclos}{Frec.reloj}$$

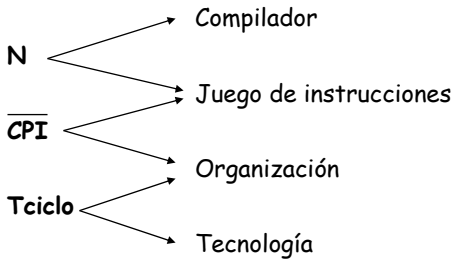
$$T.ejecución = Nº\,instrucciones \bullet CPI \bullet Tciclo$$

Parámetros de prestaciones

El tiempo de ejecución como medida.

$$\overline{T} = N \bullet \overline{CPI} \bullet Tciclo$$

Objetivo: Reducir cuaquiera de los tres factores.



Parámetros de prestaciones

Productividad

$$MIPS = \frac{N^{o}\, instrucciones}{T.ejecución \bullet 10^{6}}$$

» **MIPS** de pico, MIPS relativos y **MFLOPS**

$$MIPS_{relativos} = \frac{T.ejecucion_{Mreferencia}}{T.ejecución_{Mx}} \bullet MIPS_{Mreferencia}$$

Parámetros de prestaciones

Ejemplo: Cálculo de MIPS. (Tciclo = 1ns)

Nº instrucciones ($x10^9$) de tipo:

Código	A (5 ciclos)	B (6 ciclos)	C (7 ciclos)
M1	3	2	1
M2	7	1	1

T.ejecución (M1) = (15+12+7) x 10^9 x 1 x 10^{-9} = **34s**
T.ejecución (M2) = (35+6+7) x 10^9 x 1 x 10^{-9} = 48s

MIPS (M1) = $(6x10^9)/34x10^6$ = 176,47
MIPS (M2) = $(9x10^9)/48x10^6$ = **187,5**

Parámetros de prestaciones

Utilización de *benchmarks* (SPECs)

"Standard Performance Evaluation Corporation"

$$SPEC_{ratio} = \frac{T.ejecución_{SunUltra5-10}}{T.ejecución_{Mx}}$$

SPEC Benchmark suite

Conjunto de benchmarks elaborados a partir de aplicaciones científicas y de ingeniería.

CPU: spec cpu2000 (cint2000, cfp2000)

Spec cpu2000 (MPR dic-2003)

Processor	Alpha 21364	AMD Opteron	HP PA-8700	IBM Power 4+	Intel Itanium 2	Intel XeonMP	Intel Xeon	MIPS R14000	Sun UltraSPARC III
System or Motherboard	Alpha GS1280/7	ASUS SK8N	HP9000 C3750	pSeries 650 6M2	HP RX2600	Dell PwrEdg 6650	Dell Prec. 650	SGI 3200	Sun Fire 280R
Clock Rate	1.15GHz	2.2GHz	870MHz	1.7GHz	1.5GHz	2.8GHz	3.2GHz	600MHz	1.20GHz
External Cache	None	None	None	128MB	None	None	None	8MB	8MB
164.gzip	583	1,203	588	799	973	1,089	1,254	322	474
175.vpr	822	1,215	688	1,053	1,119	959	930	572	544
176.gcc	859	1,430	906	1,141	1,554	1,452	1,654	445	674
181.mcf	712	1,084	494	1,804	2,305	882	821	783	684
186.crafty	982	1,504	751	944	1,221	1,096	1,233	502	660
197.parser	514	1,355	495	453	983	1,125	1,226	409	565
252.eon	958	1,597	592	1,344	1,547	1,236	1,418	507	762
253.perlbmk	768	1,461	619	828	1,210	1,262	1,344	367	636
254.gap	636	1,552	339	1,115	1,074	1,416	1,655	308	462
255.vortex	1,094	2,182	1,196	1,663	1,944	1,794	2,040	679	992
256.bzip2	824	1,185	534	1,130	1,217	1,024	1,001	493	691
300.twolf	1,018	1,368	911	1,408	1,280	1,383	1,187	645	652
SPECint_base2000	795	1,405	642	1,077	1,322	1,201	1,274	483	637
168.wupside	883	1,674	446	2,220	1,466	1,287	1,606	434	888
171.swim	3,590	2,254	931	2,140	4,136	1,308	1,562	529	1,023
172.mgrid	708	1,254	621	1,041	2,503	1,025	1,167	379	700
173.applu	1,518	1,344	702	1,393	3,469	845	1,093	381	741
177.mesa	928	1,603	694	856	1,126	1,137	1,315	425	693
178.galgel	2,105	2,679	1,603	3,794	4,225	2,291	2,016	1,398	1,977
179.art	2,014	1,468	670	2,314	6,514	1,489	901	1,436	9,951
183.equake	519	1,422	413	2,950	2,982	1,024	1,354	347	1,537
187.facerec	1,105	2,016	430	1,928	1,925	1,253	1,493	647	1,172
188.ammp	735	1,299	553	1,046	1,413	977	867	573	592
189.lucas	1,522	1,529	448	1,834	1,774	1,153	1,354	442	520
191.fma3d	1,019	1,438	404	1,290	1,091	939	1,192	306	489
200.sixtrack	469	684	471	724	1,402	525	590	298	406
301.aspi	1,242	1,366	696	1,345	1,024	979	1,003	406	644
SPECfp_base2000	1,124	1,505	600	1,598	2,119	1,103	1,200	499	945

Tendencias arquitectónicas

Tendencia clásica: Arquitecturas CISC

- Reducción del tamaño de los programas.
- Aumento del CPI (instrucciones complejas y compactas).
- Unidad de control microprogramada.
- Compiladores complejos que no siempre aprovechan todo el repertorio de instrucciones.

Tendencia actual: Arquitecturas RISC

- Aumento del tamaño de los programas.
- Reduce el CPI (menor complejidad ruta de datos).
- Instrucciones sencillas (control cableado).

Mecanismos utilizados

Memorias cache.
Modelo de ejecución registro-registro.
ALUs especializadas.

Reducción de Latencia

Paralelismo temporal (*pipeline*):

- memoria
- procesador

Paralelismo espacial:

- memoria
- procesador

Aumento de productividad (throughput)

Efectividad de la mejora

Ganancia (*Speedup*):

$$Ganancia = \frac{T.ejecucion_{\ original}}{T.ejecución_{\ nuevo}}$$

Ley de Amdahl:

La ganancia que se puede obtener introduciendo una mejora en algún componente del computador, está limitada por la fracción del tiempo total en que ésta no se utiliza.

$$Tiempo_{nuevo} = T_{original} \bullet \left[(1 - Fraccion_{\ mejorada}) + \frac{Fraccion_{mejorada}}{Ganancia_{mejorada}} \right]$$

$$Ganancia_{total} = \frac{T_{original}}{T_{nuevo}} = \frac{1}{(1 - Fraccion_{mejorada}) + \frac{Fraccion_{mejorada}}{Ganancia_{mejorada}}}$$

Efectividad de la mejora

Ejemplo

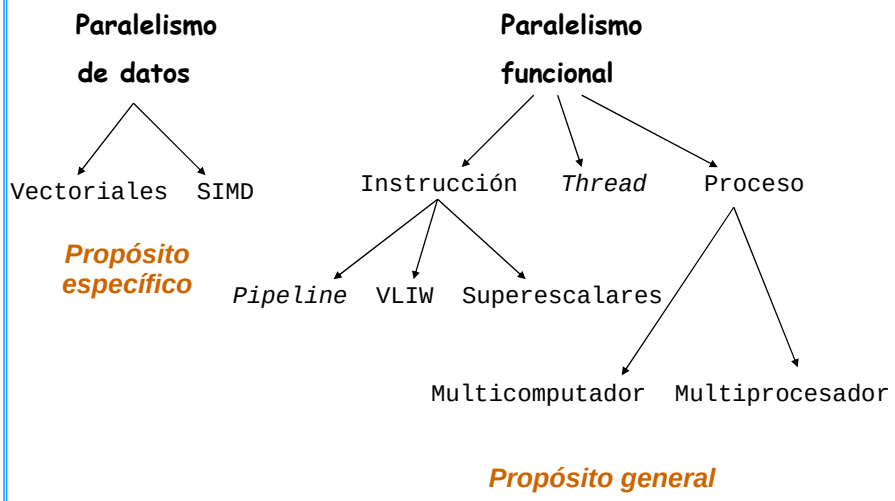
Ganancia obtenida mejorando el comportamiento de las instrucciones de coma flotante.

- $Ganancia_{mejorada}$ = **5**
- T. $Ejecución_{original}$ = **20 s**
- $Fracción_{mejorada}$ = **0,5**

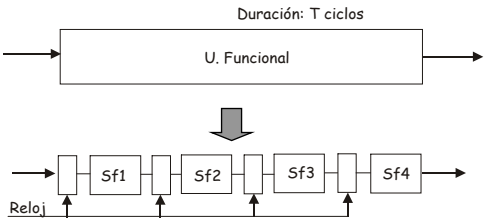
Tiempo$_{nuevo}$ = **12 s**

Ganancia$_{total}$ = **1,66**

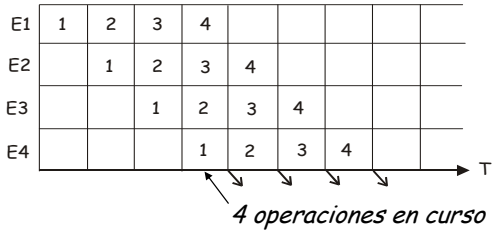
Arquitecturas paralelas



Segmentación (*Pipeline*)



Objetivo: Aumentar la productividad ⇨ Posible Incremento de latencia



- Máxima efectividad:
  T/4 ciclos por etapa = **T. ciclo**
- Productividad *4
- Aceleración ideal:
  Nº de etapas

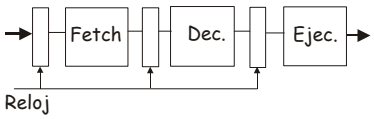
Pipeline de instrucciones

Permite solapar la ejecución de varias instrucciones.

Objetivo:

Aumentar la productividad de la CPU (nº instrucciones/s)

- Ejecución de instrucciones: Fases que se realizan secuencialmente.
- **Diseño con *pipeline*:**

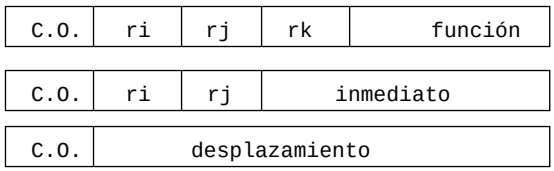


Ciclo de máquina: T. necesario para pasar de una etapa a la siguiente.
Duración determinada por la etapa mas lenta.

Máxima efectividad si todas las etapas tardan el mismo tiempo (Mca).

Pipeline de instrucciones: Caso de estudio

Arquitectura RISC. Modelo de ejecución registro-registro

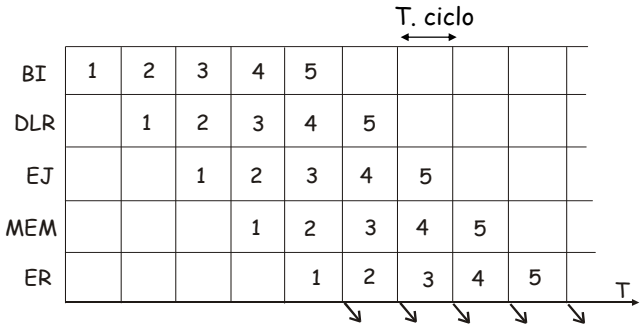


Formatos de Instrucción (1 palabra)

Etapas:

BI: Acceso a McaI; PC+4 -> PC
DLR: Decodificación y Lectura de registros
EJ: Opalu/cálculo dirección; Comprobación cond. salto.
MEM: Acceso a McaD/Actualización del PC si se salta
ER: Escritura en registros.

Pipeline de instrucciones: Caso de estudio



Productividad sin *pipeline*: 1 instrucción cada 5 ciclos

Productividad ideal con *pipeline*: 1 instrucción por ciclo.

Ganancia ideal = 5

Pipeline de instrucciones

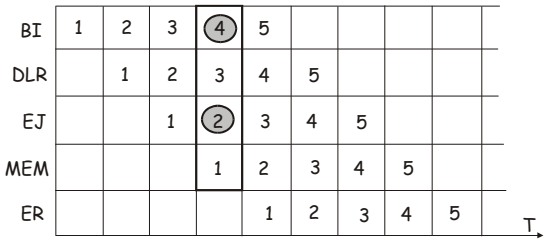
Aspectos de diseño.

- Ajuste de los tiempos de duración de cada etapa.
- Acciones a realizar en cada etapa

⇨ Influencia en el hardware

Consideraciones:

1) El PC se debe incrementar cada ciclo, en BI.



⇨ Incrementador especial

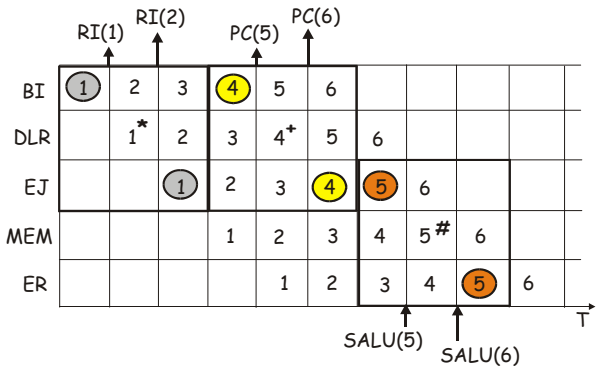
Pipeline de instrucciones

2) **Se necesita conservar información** que se utilizará posteriormente.

a) El Registro de Instrucción (campos inmediato y desp.)

b) El PC (para saltos relativos)

c) La salida de la ALU (para almacenar en registros)




```
Ej:
   ld r1, #n[r2]
....
   br $desp
   add r1,r2,r3
   sub r4,r5,r6
```

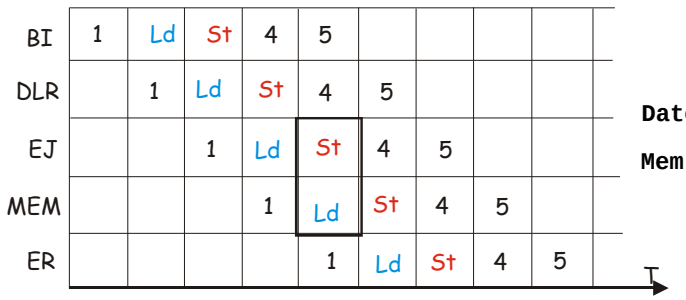
* RI' ← RI

\+ PC' ← PC

\# SALU' ← SALU

Pipeline de instrucciones

3) **Se necesitan registros distintos** para *load* y *store*.



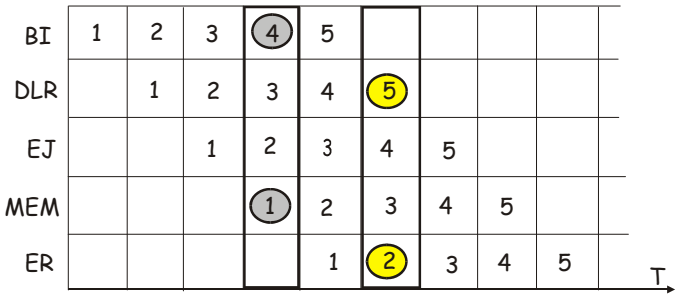
Dato -> SRDAT

Mem -> LRDAT

Pipeline de instrucciones

4) **Se necesita** dar soporte a **2 accesos a memoria en el mismo ciclo.**
Arquitectura *Harvard*

5) **Se necesitan 2 accesos al banco de registros en cada ciclo.**
Puertos de lectura y escritura



Pipeline de instrucciones

Registros de pipeline

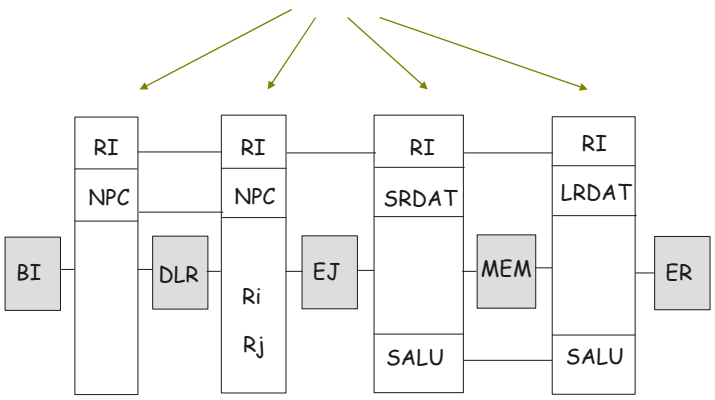
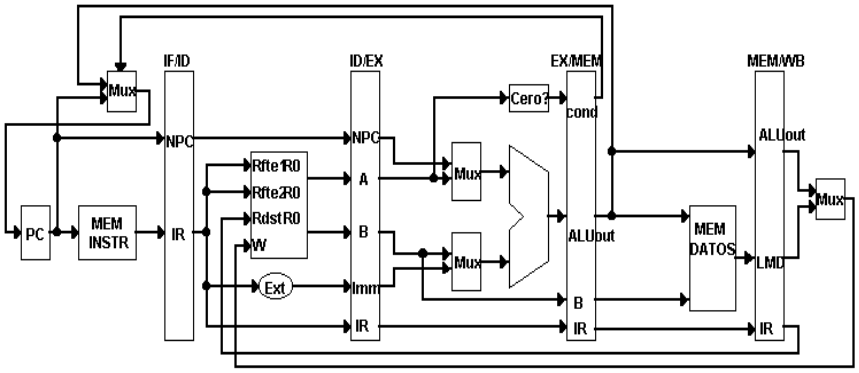


Diagrama de bloques

BI DLR EJ MEM ER



Pipeline de instrucciones

Si CPI (sin pipeline) = 5
Y CPIideal (con pipeline) =1

Incremento de productividad = 5

¿Se puede alcanzar?

Hay situaciones que impiden que en cada ciclo se inicie la ejecución de una nueva instrucción: **DEPENDENCIAS**

-> **burbujas** en el pipeline (*stalls*)

-> **pérdida de prestaciones**

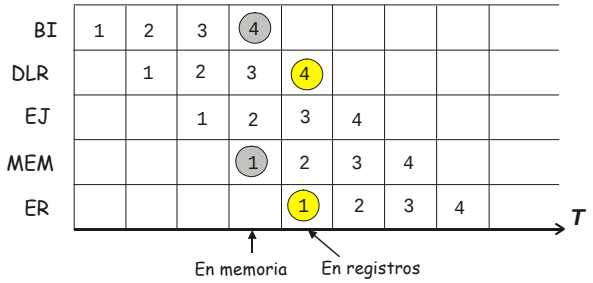
CPIreal = CPIideal + (nº burbujas/instrucción) > 1

Tipos de Dependencias

- Estructurales
- De datos
- De control

Dependencias estructurales

Ejemplos típicos:

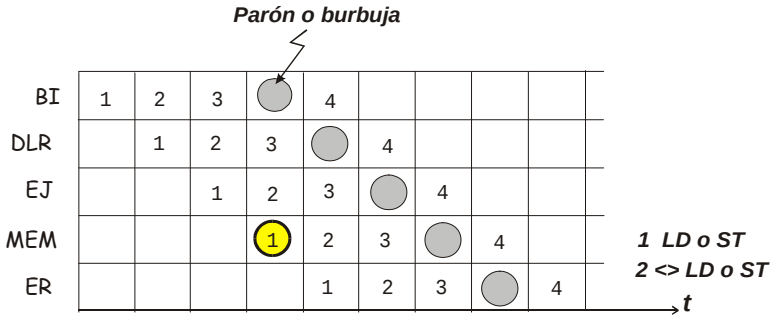


Causa: Un recurso no se ha replicado suficientemente.

Tipos de Dependencias

Efectos de las dependencias estructurales

Ejemplo: Cache unificada



¿Impacto en CPI si 40% ld/st?

Tipos de dependencias de datos

-RAW (*read after write*)

add **r1**,r2,r3
sub r4,**r1**,r5

-WAR (*write after read*)

add r1,**r2**,r3
sub **r2**,r4,r5

-WAW (*write after write*)

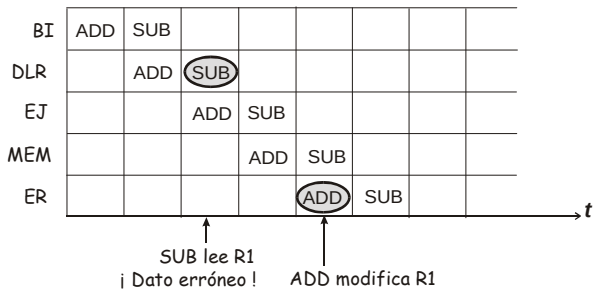
add **r1**,r2,r3
sub **r1**,r4,r5

Tipos de Dependencias

Dependencias de datos (RAW)

Una instrucción depende de los resultados producidos por otra anterior.

Ejs: **add r1, r0,#50** **ld r1, #2[r2]**
sub r2, r1, r3 **add r3, r1, r4**



Soluciones a las dependencias RAW

1. Ciclos de espera.
2. Anticipación (*forwarding*).
3. Reordenación de código.

1. Introducción de ciclos de espera

- ***Software*** (instrucciones NOP):

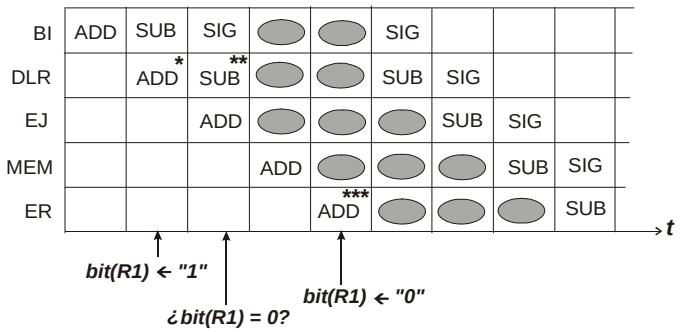
Ej:

```
add r1, r0,#50
Nop
Nop
Nop
sub r2, r1, r3
```


Soluciones a las dependencias RAW

1. Introducción de ciclos de espera

- ***Hardware***. Detección automática: **Marcador** *(Scoreboarding)*



Soluciones a las dependencias RAW

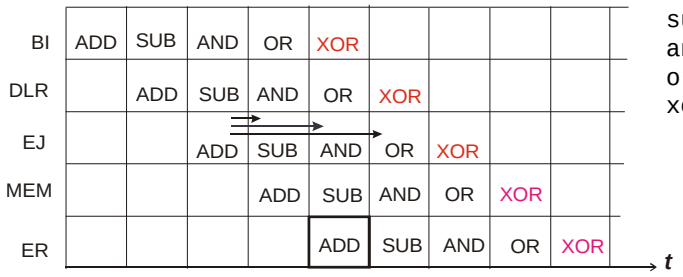
2. Anticipación (*forwarding*)

Un resultado temporal se envia **directamente** desde la unidad que lo genera a la entrada de la que lo necesita.

- **Anticipación ALU -> ALU**

Ej:

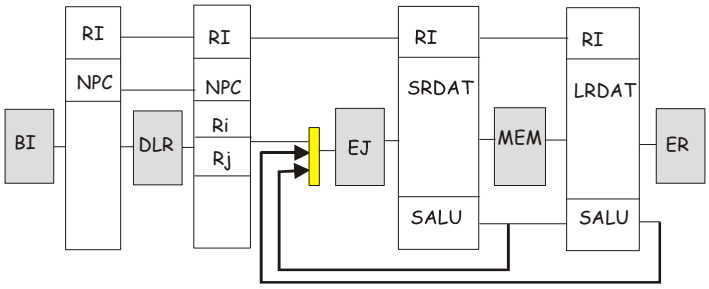
```
add r1,r2,r3
sub r4,r1,r5
and r6,r1,r7
or r8,r1,r9
xor r10,r1,r11
```



Soluciones a las dependencias RAW

Anticipación ALU -> ALU (2 niveles)

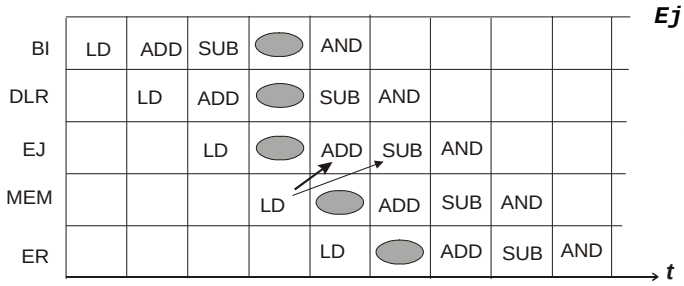
- Caminos adicionales + multiplexores



Soluciones a las dependencias RAW

2. Anticipación (*forwarding*)

- **Anticipación MEM -> ALU**



Ej:

```
ld r1,#0[r6]
add r4,r1,r7
sub r5,r1,r8
and r6,r1,r7
```

3 ciclos de espera ⟹ 1 solo ciclo

Soluciones a las dependencias RAW

3. Reordenación de código.

Ej:

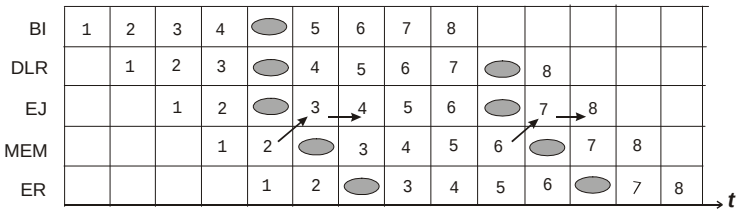
1. ld **r2**, b
2. ld **r3**, c
3. add **r1**,**r2**,**r3**
4. st **r1**, a
5. ld r5, e
6. ld r6, f
7. sub **r4**,r5,r6
8. st **r4**, d

$$a = b + c$$
$$d = e - f$$

⟹

Reordenado

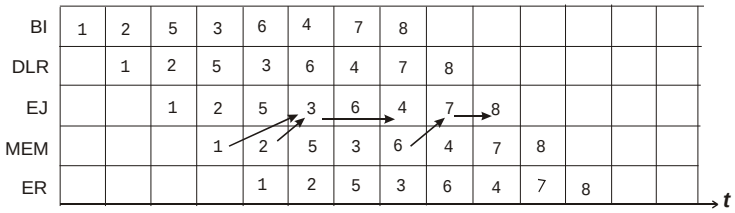
ld r2, b
ld r3, c
ld r5, e
add r1,r2,r3
ld r6, f
st r1, a
sub r4,r5,r6
st r4,d



Soluciones a las dependencias RAW

Código reordenado:.

```
1    ld r2, b
2    ld r3, c
5    ld r5, e
3    addr1,r2,r3
6    ld r6, f
4    st r1, a
7    sub r4,r5,r6
8    st r4,d
```

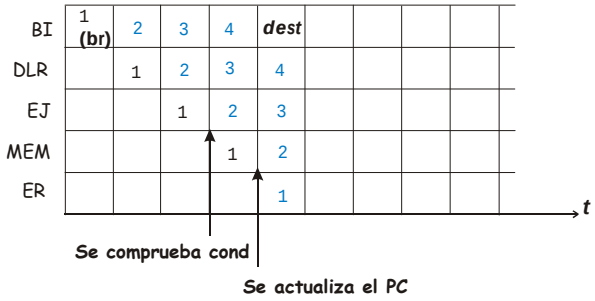


Tipos de dependencias

Dependencias de control.

Instrucciones que pueden alterar el flujo secuencial de ejecución.

- **Saltos condicionales e incondicionales**



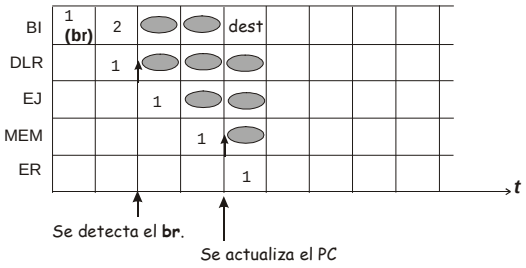
¡Han comenzado ya las instrucciones secuenciales al salto!

Dependencias de control

Solución 1.

Insertar **ciclos de espera** (*hardware* o *software*).

Solución *hardware:*



3 parones por cada instrucción *br* ⟹ ***Pérdida de prestaciones***

¿% de instrucciones de salto?

- Aplicaciones de propósito general ≈ 20%
- Aplicaciones de cálculo científico ≈ 5-10%

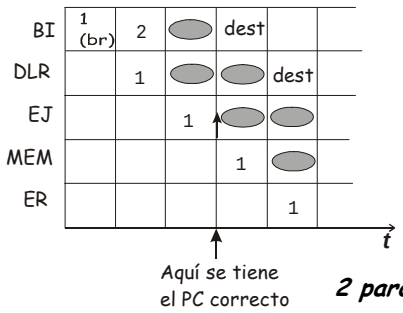
Dependencias de control

OBJETIVO: Reducir el número de ciclos de espera.

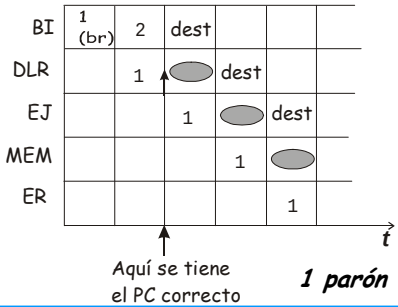
- Averiguar lo antes posible si el salto es efectivo (brcc).
- Calcular lo antes posible la dirección de salto.

Posibles alternativas:

1) Hacer ambas cosas **en EJ**. **2)** Hacer ambas cosas **en DLR**



2 parones

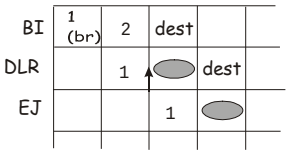


1 parón

Dependencias de control

Requisitos de la alternativa 2:

- Sumador para calcular la dirección
- Circuito para comprobar la condición.



Aumenta la **complejidad y la duración de la etapa DLR**.

Las **condiciones** de salto deben ser **sencillas**:

Ejemplos:

- **bz r2, $direccion**
- **bb0 y bb1 del mc88110**

1 parón

Dependencias de control

Solución 2. Saltos retardados (*software*).

Ocupar los huecos tras los saltos con instrucciones cuya ejecución sea válida tanto si se salta como si no.

Branch delay slots: Número de instrucciones a colocar detras de cada salto.

Número de ciclos que transcurren desde que se realiza el fetch de la instrucción "br" hasta que se realiza el *fetch* de la instrucción a la que se salta.

Estrategias (1 *delay slot*):

a) Con alguna instrucción, independiente, anterior al salto.
b) Con la instrucción destino del salto.
c) Con la instrucción siguiente a la de salto.

Ejemplos de saltos retardados

a)

```
add r1,r2,r3          bz r2, $dir
bz r2, $dir           add r1,r2,r3
  hueco
```

b)

```
dir:  sub r4,r5,r6        dir:   sub r4,r5,r6
      and r7,r8,r0        dir1:  and r7,r8,r0
      .........                  ..........
      add r1,r2,r3               add r1,r2,r3
      bz r1, $dir                bz r1, $dir1
        hueco                    sub r4,r5,r6
```

c)

```
      add r1,r2,r3          add r1,r2,r3
      bz r1, $dir           bz r1, $dir
        hueco               sub r4,r5,r6
      sub r4,r5,r6                ..............
      ............          dir:  and r7,r8,r0
dir:  and r7,r8,r0
```


Dependencias de control

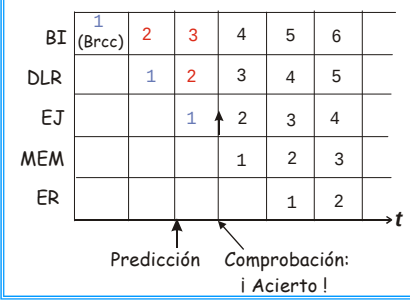
Solución 3. Predicción de salto (saltos condicionales)

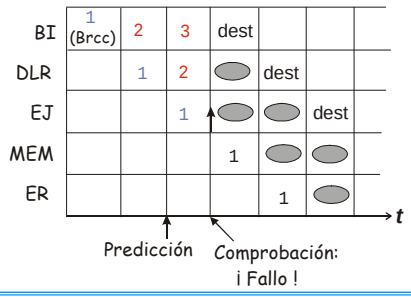
% de saltos condicionales ≈ 80%

Suponer a priori un valor de la condición, y una vez evaluada:

- Si se ha acertado, continuar.
- Si se ha fallado, desechar la instrucción o instrucciones.

Ejemplo: **Predicción NO SALTO**





Predicción de salto

Requisitos hardware:

- Detección del error de predicción
- Registros adicionales para desechar instrucciones.

TIPOS DE PREDICCIÓN.

Fija.	Estática	Dinámica
Siempre la misma apuesta	basada en el código	basada en la historia de la ejecución.

Prestaciones y complejidad de implementación

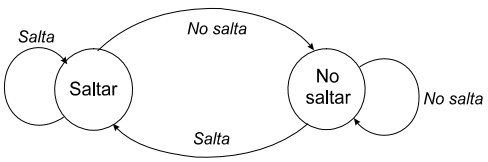
Predicción dinámica:

Si el salto **fue efectivo** en la(s) **última**(s) **ejecución**(es), es muy **probable** que lo sea en la **siguiente**.

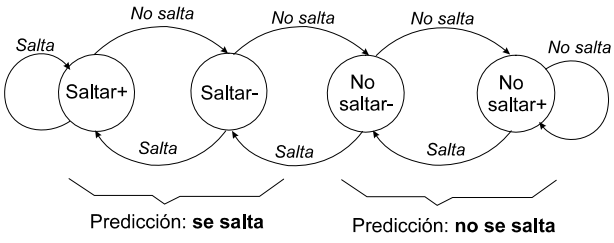
Predicción dinámica de salto

Hay que **almacenar** y **actualizar** la historia **de las instrucciones de salto condicional.**

- **1 bit:**



- **2 bits:**



Dependencias de control

Unrolling

```
for (i=0; i< N; i++)
   r = r + a[i] + b[i]
```

```
for (i=0; i< N; i+=2)
  { r = r + a[i] + b[i];
    r = r + a[i+1] + b[i+1]; }
```

```
buc:  ld r5, #0[r1]
      ld r6, #0[r2]
      add r7, r5, r7
      add r7, r6, r7
      add r1, r1, #4
      add r2, r2, #4
      sub r4, r4, #1
      bnz r4, $buc
```

```
buc:  ld r5, #0[r1]
      ld r6, #0[r2]
      add r7, r5, r7
      add r7, r6, r7
      ld r8, #4[r1]
      ld r9, #4[r2]
      add r7, r8, r7
      add r7, r9, r7
      add r1, r1, #8
      add r2, r2, #8
      sub r4, r4, #2
      bnz r4, $buc
```

Reducción del nº de instrucciones de salto